Software Testing

PERCORSI PER LE COMPETENZE TRASVERSALI E PER L'ORIENTAMENTO

ANNO SCOLASTICO 2023-2024

– Ing. Porfirio Tramontana

e-mail: porfirio.tramontana@unina.it

http://wpage.unina.it/ptramont/

Calendario

- **8 febbraio 2024 : 9.30 – 14.00, aula II-3, Edificio 1 Via Claudio**
- **21 febbraio 2024 : 9.30 – 18.00 , aula II-3, Edificio 1 Via Claudio**
- **29 febbraio 2024 : 13.30 – 18.00 , aula II-3, Edificio 1 Via Claudio**
- **1 marzo 2024 : 9.30 - 14.00 , aula II-3, Edificio 1 Via Claudio (da confermare)**

Materiale da scaricare

- https://sites.google.com/view/porfiriotramontana/pcto2024

- **Eclipse download : https://www.eclipse.org/downloads/**
  - Insieme ad Eclipse c'è anche una versione di Java e Junit, che saranno gli strumenti fondamentali che utilizzeremo
- **Selenium : https://chromewebstore.google.com/detail/selenium-ide/mooikfkahbdckldjjndioackbalphokd**
  - Selenium è un plug-in disponibile per tutti i browser cje consente di creare casi di test

Di cosa ci occuperemo …

- **Testing del software**
  - Che cosa è? A cosa serve? Ma serve davvero? Ma perché si fa? Chi lo fa? Chi lo insegna? Come si può fare? Come si fa a vedere se è riuscito bene?
- **Casi di test**
  - Cosa sono? Come si definisce un test? I test possono essere utilizzati anche al di fuori del software?
- **Testing automation**
  - I test possono essere eseguiti automaticamente dal computer? Il computer sa riconoscere se il test è andato bene oppure no? Il computer può creare da solo i casi di test?

Di cosa ci occuperemo …

- **Testing Black Box**
  - Possiamo provare un programma senza sapere come è fatto al suo interno? Come facciamo a capire se ha funzionato bene? Come facciamo a capire se abbiamo fatto abbastanza prove?
- **Testing White Box**
  - Come possiamo testare meglio un programma se sappiamo come è stato scritto? Come facciamo a capire se ha funzionato bene? Come facciamo a capire se abbiamo fatto abbastanza prove?

In aggiunta …

- **Cosa si insegna ad Ingegneria Informatica relativamente al software?**
- **Cosa si insegna ad Informatica relativamente al software?**
- **Come sono strutturati questi corsi di laurea?**

Tipologia di lezione

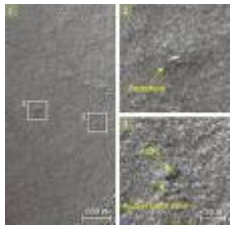
- **Un po' di teoria e definizioni, direttamente dalle slide, ma ragionandoci un po'**
- **Un po' di esercizi da risolvere con carta e penna o sul mio pc**
- **Un po' di esercizi da risolvere con il pc**
- **Un po' di esercizi da risolvere a casa**

- **Esercizi di progettazione dei casi di test, ricerca degli errori, correzione degli errori e altro**

Motivazioni

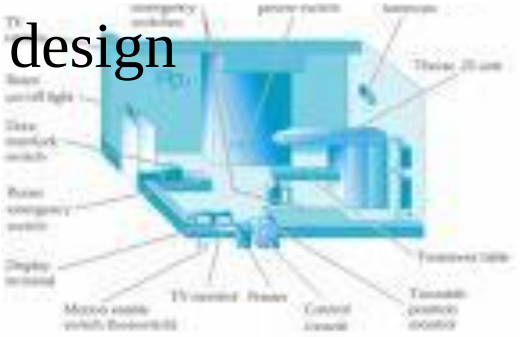
Spectacular Software Failures

- NASA's Mars lander: September 1999, crashed due to a units integration fault



Mars Polar Lander crash site?

THERAC-25 design



- THERAC-25 radiation machine : Poor testing of safety-critical software can cost *lives* : 3 patients were killed
- Ariane 5 explosion : Millions of $$
- Intel's Pentium FDIV fault : Public relations nightmare

Ariane 5: exception-handling bug : forced self destruct on maiden flight (64-bit to 16-bit conversion: about 370 million $ lost)

We need our software to be dependable

Testing is *one* way to assess dependability

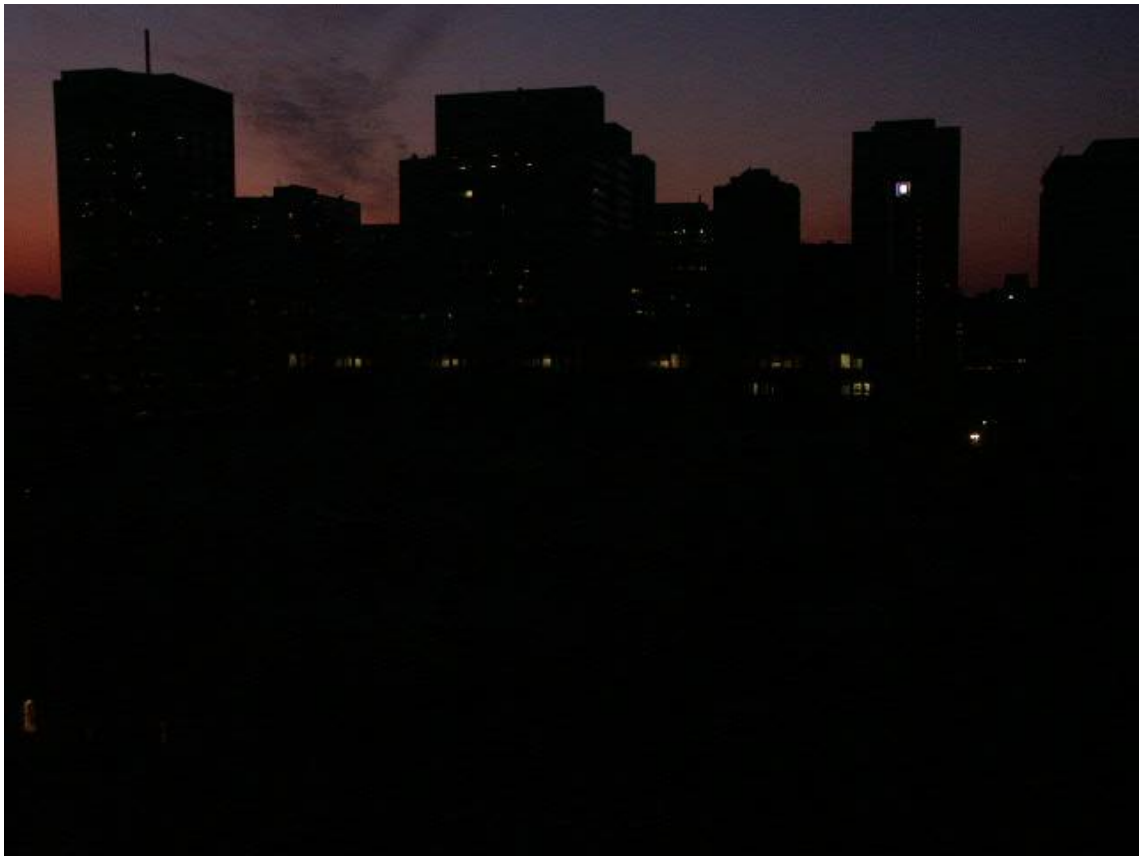
Northeast Blackout of 2003

508 generating units and 256 power plants shut down

Affected 10 million people in Ontario, Canada

Affected 40 million people in 8 US states

Financial losses of $6 Billion USD



The alarm system in the energy management system failed due to a software error and operators were not informed of the power overload in the system

Costly Software Failures

- NIST report, "The Economic Impacts of Inadequate Infrastructure for Software Testing" (2002)
  - Inadequate software testing costs the US alone between $22 and $59 billion annually
  - Better approaches could cut this amount in half
- Huge losses due to web application failures
  - Financial services : $6.5 million per hour (just in USA!)
  - Credit card sales applications : $2.4 million per hour (in USA)
- In Dec 2006, *amazon.com's* BOGO offer turned into a double discount
- 2007 : Symantec says that most security vulnerabilities are due to faulty software

Spectacular software Failures

- Boeing A220 : Engines failed after software update allowed excessive vibrations

- Boeing 737 Max : Crashed due to overly aggressive software flight overrides (MCAS)
- Toyota brakes : Dozens dead, thousands of crashes

- Healthcare website : Crashed repeatedly on launch—never load tested
- Northeast blackout : 50 million people, $6 billion USD lost … alarm system failed

Software testers try to find faults before the faults find users

Cost of Not Testing

Poor Program Managers might say:
"Testing is too expensive."

- **Testing is the most time consuming and expensive part of software development**
- **Not testing is even more expensive**
- **If we have too little testing effort early, the cost of testing increases**
- **Planning for testing after development is prohibitively expensive**

Testing

Theoretical Foundations

Verify and Validation (V&V)

- Verify and Validation (V & V) show is the software satisfies its requirements and (Verification) and if it satisifies user requirements (Validation).
  - *Verification: Are we making the product right?*
  - *Validation: Are we making the right product?*
- Two complementary approaches to the verification:
  - Experimental approach (by testing or dynamical analysis of the software during its execution)
  - Analytic approach (static analysis, of code and documentation, formal analysis)

Definitions, in better detail

- **Error** (human error)
  - Human error in programming, due to wrong interpretations, insufficient knowledge of the problem to solve, material mistake or any other issue
- **Defect** (fault, bug)
  - Defect into the software code due to a human error, that could cause the failure of the software system
- **Failure**
  - An execution of the software that does not provide the expected results and behaviour
  - Failures may be dynamic: they happens in a given time due to a given input and can be observed only via software execution

An example

```
void raddoppia()
{
cin>>x;
y := x*x;
cout<<y;
}
```

An example

```
void raddoppia()
{
cin>>x;
y := x*x;
cout<<y;
}
```

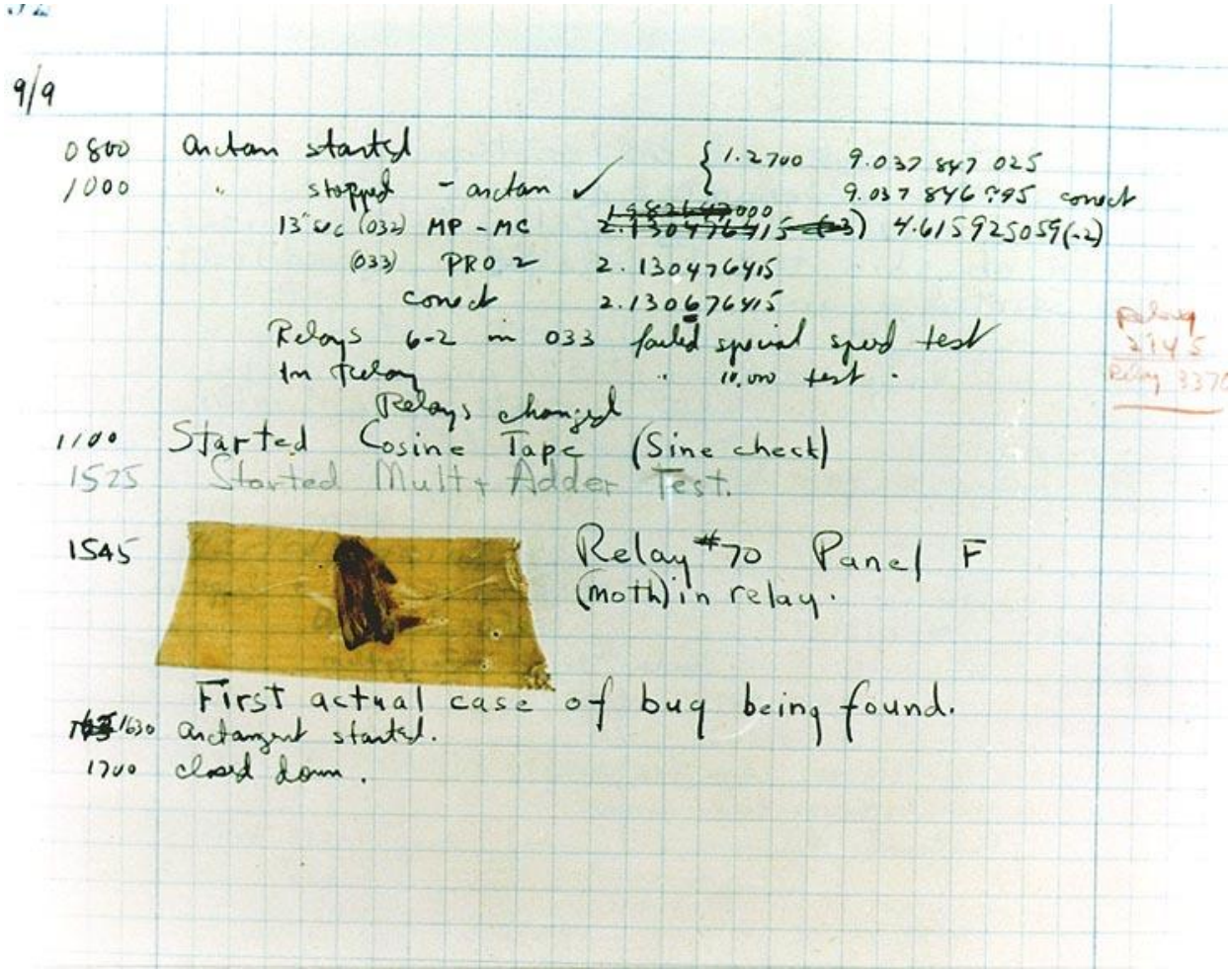
ERROR due to lack of knowledge or editing

ERROR
We think that the double of x can be computed as x*x OR we have typed * instead of +

FAULT
“*” instead of “+”

FAILURE
=> The wrong result is shown

The failure can be detected by inputting 10 (but it could not be detected by inputting 2 …)



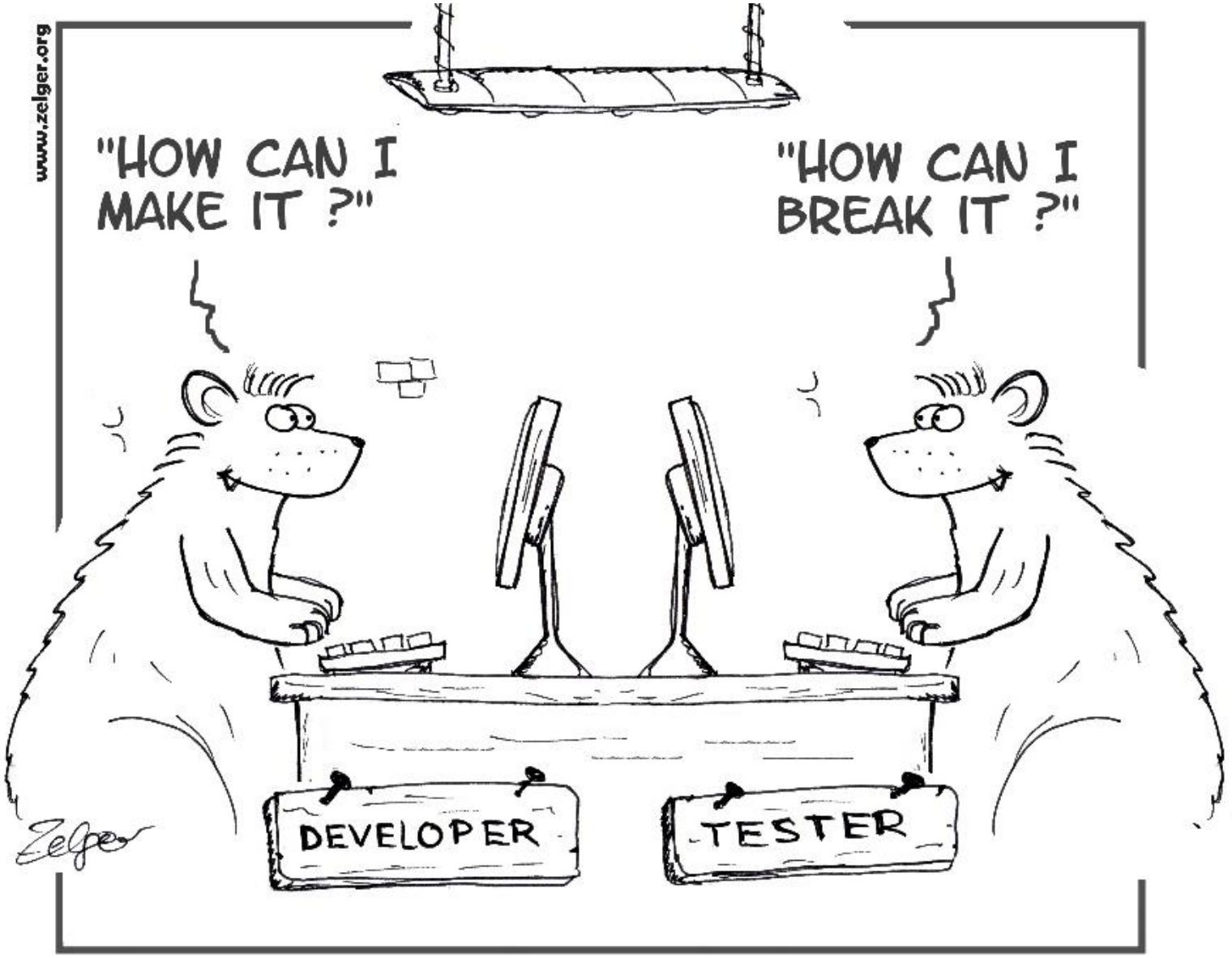
Grace Hopper's bug

- Note: probably the first use of the 'bug' term as a fault is due to Thomas Edison

Defect testing

- It is used to find faults in a software
  - Faults are found by searching for failures
- Defect testing *succeeded* if the program fails (i.e. failures are observed)
  - Why testing *fails* if no defects are found? It is a philosophical question

- Testing may demonstrate the presence of faults
  - May testing demonstrate the *absence* of faults, too?



They weren't so much different, but they had different goals

Io non volevo solo partecipare alle feste. Volevo avere il potere di farle fallire.

Geppino «Jep» Gambardella

Legge di Murphy

Se qualcosa può andar male, lo farà

Dijkstra Thesis

- **Dijkstra Thesis**
  - Testing cannot prove absence of defects, but only their presence
- There is no guarantee that if at the n-th test a module or a system has responded correctly (i.e. no defects have been found), the same can be done at the (n+1)-th test
- Inability to produce all possible configurations of input values (test cases) at all possible internal states of a software system

- **"We did about 10,000 tests on it, and it was working fine until Monday."**
  - Anonymous - Spokesperson for 7-11 after Y2K-related failure of their credit card processing on 2001-01-01

Absence of continuity properties

- In many fields of engineering, testing is simplified by the existence of **continuity** properties.
  - If a bridge withstands a load of 1000 tons, then it will withstand even lighter loads
- In the field of software we are dealing with **discrete** systems, for which small variations in input values can lead to incorrect results
  - Exhaustive testing (ideal) is a necessary condition to be able to evaluate the correctness of a program starting from testing

Legge di Murphy

Se qualcosa può andar male, lo farà

Corollari

- Niente è facile come sembra.

Dependency on input values

```
char x;
cin>>x;
```

In how many ways can be executed this code?

x is a char (with 256 possible values), so the code can be executed in 256 different ways but …

the >> operator allows the insertion of unlimited input sequences:

abcde0

qiunonooiiouuoiounoinoinoinonoiioio0

‘string causing buffer overflow’

0

...

Amenità sul Testing

da "The Zen of Programming" - Geoffrey James

Thus spoke the master: "Any program, no matter how small, contains bugs"

The novice did not believe the master's words. "What if the program were so small that it performed a single function?" he asked.

"Such a program would have no meaning," said the master, "but if such a one existed, the operating system would fail eventually, producing a bug".

But the novice was not satisfied. "What if the operating system did not fail?" he asked.

"There is no operating system that does not fail," said the master, "but if such a one existed, the hardware would fail eventually, producing a bug".

The novice still was not satisfied. "What if the hardware did not fail?" he asked.

The master gave a great sigh. "There is no hardware that does not fail," said the master, "but if such a one existed, the user would want to do something different, and this too is a bug."

A program without bugs would be an absurdity, a nonesuch. If there were a program without any bugs then the world would cease to exist.

Legge di Murphy

Se qualcosa può andar male, lo farà

Corollari

- Niente è facile come sembra.
- Tutto richiede più tempo di quanto si pensi.

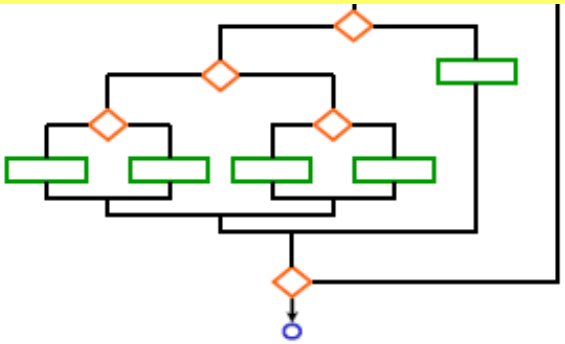
What if the program has branches and loops?

By excluding the loop, the program can be executed following 5 different paths.

By considering the loop, the program can be executed following an unlimited number of paths

By limiting the execution to n loops, the program can be executed in 5^n ways

The problem of the execution of the exhaustive test suite needs an exponential time!

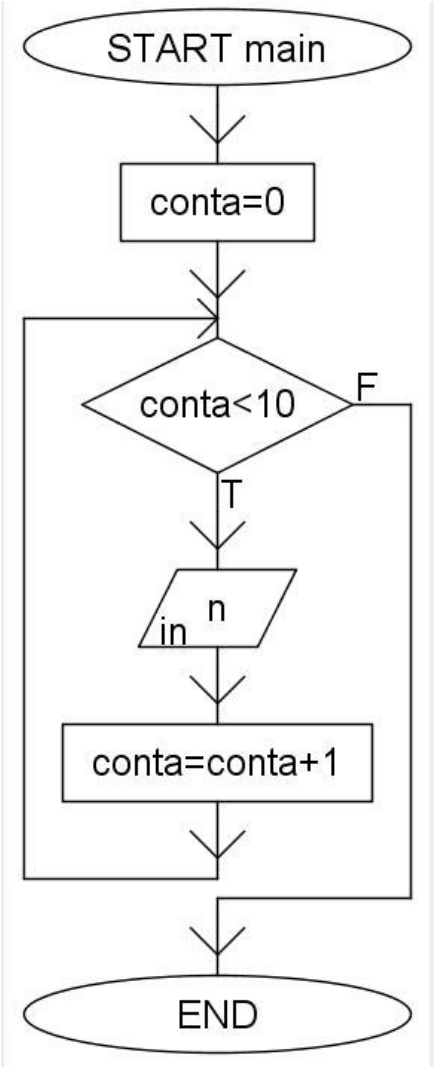


```
.....
repeat
  B0
  if R1 then
        if R2 then
                if R3 then
                        B1
                else
                        B2
                endif
                if R4 then
                        B3
                else
                        B4
                endif
              else
                B5
        endif
  endif
until R6
.........
```

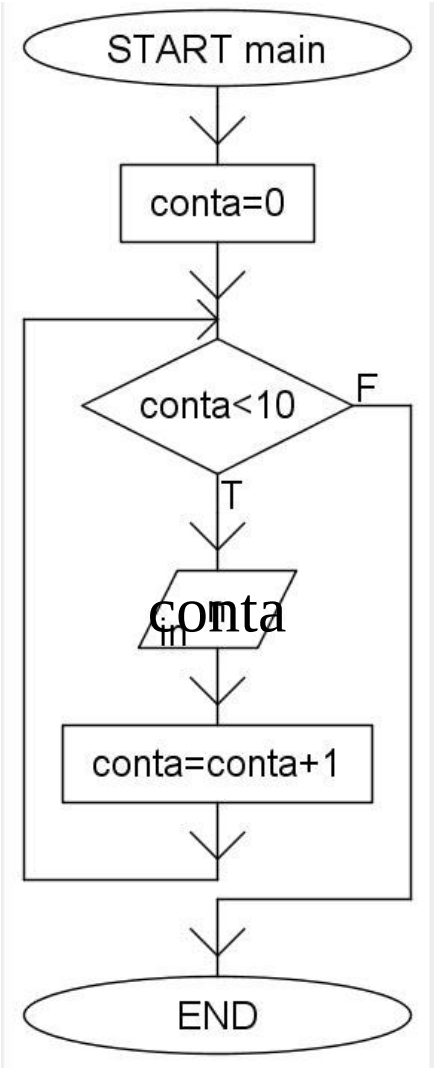
Exhaustive testing

- Exhaustive testing consists consists of the execution of all the possible behaviours of a software system
  - If exhaustive testing does not show any failure, the program is correct

- Is it always possible to generate and execute the exhaustive test suite?
  - If the program has no branches and have no inputs, then the exhaustive test suite exists and is composed of a single test case
    - *It is possible to test exhaustively the «Hello, World» program*
  - What if the program has branches and the program execution depends on input values?
  - What if the program has loops?



In quanti modi si può provare questo programma?



E quest'altro programma?

Legge di Murphy

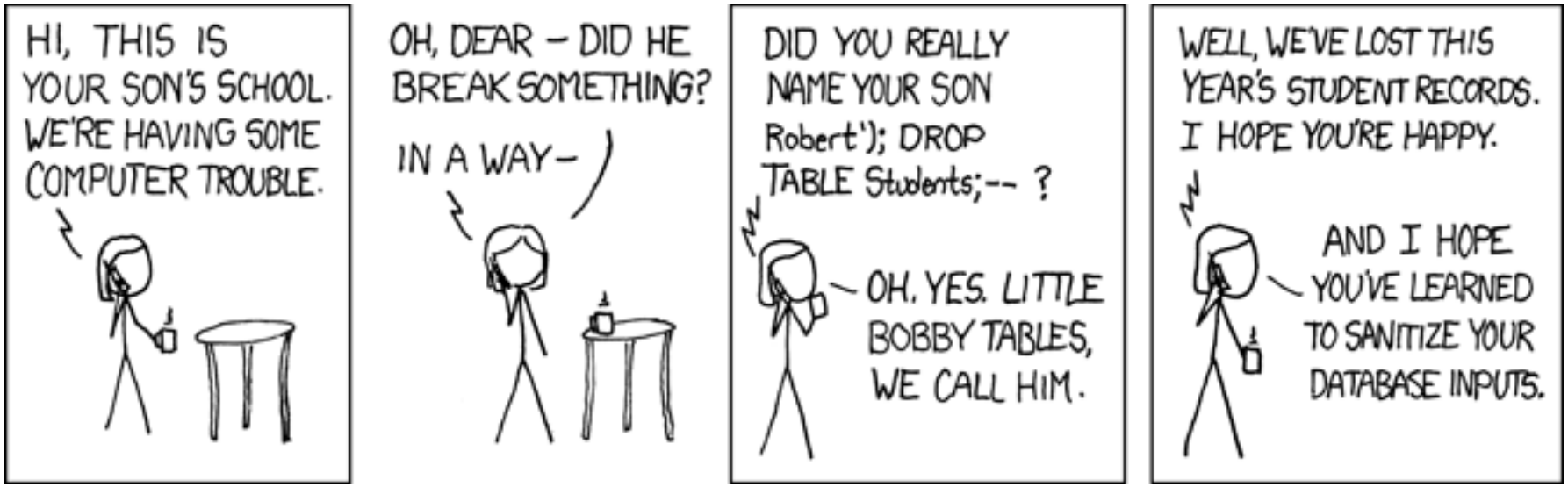
Se qualcosa può andar male, lo farà

Corollari

- Niente è facile come sembra.
- Tutto richiede più tempo di quanto si pensi.
- I cretini sono sempre più ingegnosi delle precauzioni che si prendono per impedirgli di nuocere.

"Twenty percent of all input forms filled out by people contain bad data."

- *Dennis Ritchie, More Programming Pearls: Confessions of a Coder by Jon Louis Bentley*


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

Input validation

- **A typical problem of UI testing is verifying the validity of input data**
  - It is the cause of many attacks (exploits) against applications
  - For example, in pointer-based languages, such as C, entering a string that is too long as input, in the absence of validation, can lead to overwriting other data or even areas of code of the program itself (**stack overflow problem**)
  - In interpreted languages, such as those often used for the web, the problem is even more felt

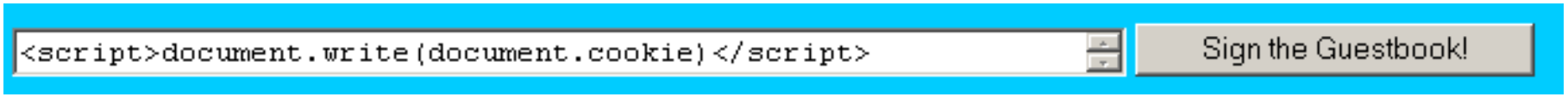
The problem of input validation into the Web

- Data validation can be done both on the client and server side
  - Client-side validation has the advantage of using client machine CPU time rather than server machine CPU time
  - However, client-side validation can be bypassed by an attacker who solicits the server with an http request that has not passed through the client page: in this case the server may have anomalies

The most correct solution is to place the validation on both the client and server side, so as to block most incorrect requests on the client (saving resources on the server). Only fraudulent requests would thus be blocked by server-side validation

Example: Cross-Site Scripting

- Occurs when a client-side script, maliciously inserted into an input field, is executed on the client machine of an unsuspecting user



password=MyPassword; login=Administrator; CurrentVersion=IT; LastVisit=20030502+10%3A36%3A17; ASPSESSIONIDAQTDQARB=HHKJLJJAONDNFOFJPLAAADDL

Sign.html

```
<form method="post" action="sign.asp">
    <textarea name="txtMessage"></textarea>
    <input type="submit" value="Sign!">
</form>
```

Message=Server.HtmlEncode(Message)

Sign.asp

```
<% Message=request.form("txtMessage")
   conn=OpenDBConnection
   set rs=server.createobject("Adodb.recordset")
   rs.open "Guestbook",conn,1,2,2
   rs.Addnew
   rs("Message")=Message
   rs.update
%>
```

Guestbook.asp

```
<% conn=OpenDBConnection
   rs=server.createobject("Adodb.recordset")
   rs.open "SELECT Message FROM GuestBook" ,
     conn,3,3
%>
<table>
<% rs.movefirst
   while not(rs.eof)
     response.write (rs.fields("Message"))
     rs.movenext
   wend
%>
</table>
<% rs.close
   set rs=nothing
   conn.close
   set conn=nothing
%>
```

The Seven Principles of Testing (from ISTQB)

- Principle 1 – Testing shows the presence of defects, not their absence
- Principle 2 – Exhaustive testing is (generally) impossible
- Principle 3 – Early testing saves time and money
  - The sooner the defect is discovered, the fewer changes need to be made to correct it
- Principle 4 – Defects tend to form clusters
  - A sort of adaptation of the principle of locality of the accesses
- Principle 5 – Beware of the pesticide paradox
  - The code "tends to adapt" to testing: in other words, to find new defects introduced over time it is always necessary to design new tests
- Principle 6 – Testing is context-dependent
  - There are no universal testing strategies and techniques applicable in every context
- -Principle 7 – The absence of errors is a false belief
  - Practical consequence of principles 1 and 2

Com’è fatto un test?

Test case specification

- Minimal set of information able to describe a test case specfication
  - ID Number and Description
  - Preconditions
    - *Hypotheses that must be veriified before the test is executed*
  - Input values
  - Expected output values (the «Oracle»)
  - Expected Postconditions
    - *Hypotheses that must be verified after the test is executed*

ID	Precond	Input	Expected Output	PostCond

Test case execution report

- In addition to the information needed for test case specification:
  - Output values
  - Result
    - *Success if preconditions and postconditions are true and output values are judged equivalent to expected output values*
    - *Failure if preconditions and postconditions are true but output values are not judged equivalent to expected output values*
    - *Not applicable (N/A) if al least a precondition or a postcondition is not true*

ID	Precond	Input	Expected Output	Output	PostCond	Result

Input

- Input can be described as attribue-value pairs …
- … or by a stream
  - For example, they can be represented by a text file
- Input may be described by a sequence of actions
  - For example in GUI based applications, input may be represented by a sequence of user actions
- Input may also be described by a set of signals and by the time of arrival to the system / exit from the system

Input

- Some examples, in order of ascending difficulty:
  - Testing of methods
    - The list of variable is known, the type of values is known
  - Testing of (web) services
    - The list of variables is known, the type of values is unknown/may be changed by the user
  - Testing of Command Line programs
    - The length of the list of variables is unknown/may be changed, the type of values is unknown/may be changed
  - Testing of GUI based programs
    - Input is composed of a list of events and input values, the length of the sequence may assume any possible value
  - Testing of videogames / real time programs
    - The (analogic) time of execution of each event / input insertion is relevant and may assume any possible value
  - Testing of distributed real time programs
    - Multiple analogic inputs may occur at different times and may assume any possible value

Example: method test case specification

- Method prototype:
  - int sum (int x, int y)

- Example of Test case Input Specifications

ID	Precond	Input	Expected Output	PostCond
		x=1 y=2		

- Input is specified as a set of pairs (name, value)

Output

- The output may be specified:
  - As values/objects in method / services testing
  - As a text on screen in command line programs
  - As a graphical screen / object on the screen in GUI program testing
  - As a combination of an output in a given time / interval in a real time system / videogame
  - …

Example: static method test case specification

- Method prototype:
  - static int sum (int x, int y)

- Example of Test case Output Specifications

ID	Precond	Input	Expected Output	PostCond
		x=1 y=2	3	

- Output is specified as a int value

Preconditions and postconditions

- Preconditions and postconditions may be about:
  - The existence and state of external services or data sources
    - *For example files, databases, services, global variables …*
  - The state of the application before/after the test execution
    - *For example: the correct authentication, the presence or absence of such data in the database, the reaching of a specific interface*
      - ***Usually, if a test is quite complex, then intermediate conditions are added to improve the fault localization ability of the test case***

Example: method test case specification

- Method prototype:
  - static int sum (int x, int y)

- Example of Test case Precondition Specification

ID	Precond	Input	Expected Output	PostCond
	None	x=1 y=2	3	

- The method has no state : no precondition are needed

Example: method test case specification

- Method prototype:
  - static int sum (int x, int y)

- Example of Test case PostCondition Specification

ID	Precond	Input	Expected Output	PostCond
	None	x=1 y=2	3	None

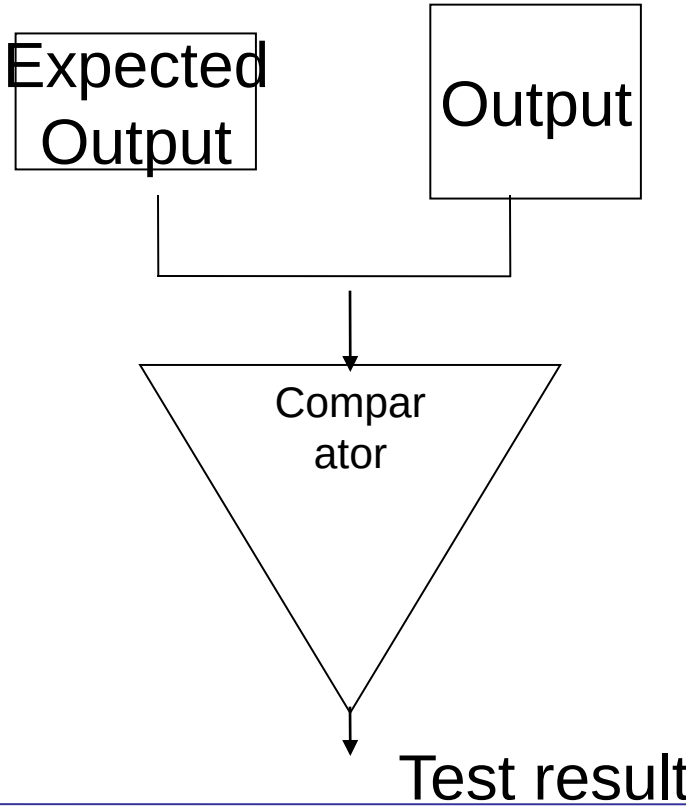
- The method has no state : no postconditions are needed

Output, Expected Output, Test Result

•The (Actual) **Output** obtained by the test execution has to be compared with the **Expected Output** designed in the test case in order to evaluate the test Result

- The **Comparator** is able to compare the oracle expected behavior and the tested software behavior and to evaluate if a failure has occurred

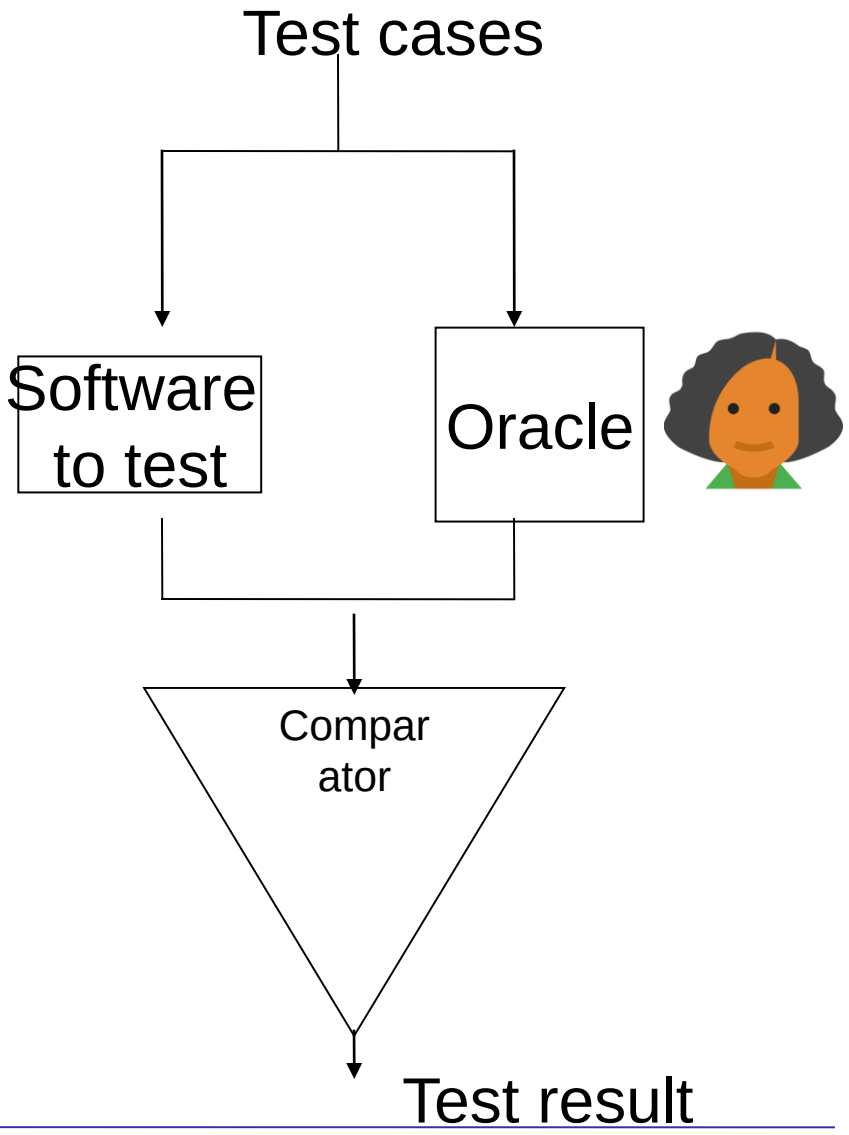
- The Comparator has to evaluate *objectively* if the Expected Output and the Output are equal / identical / equivalent between them



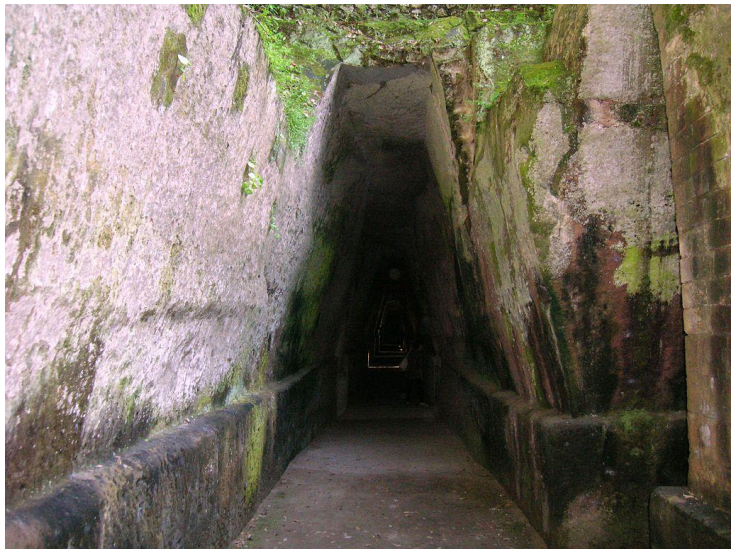
the Oracle

- the **Oracle** knows exactly what are the Expected Outputs of the system under test in response to the each test case

- In the easiest scenario, the Oracle is represented by the column of the Expected Outputs



Who is the Oracle?



- Human oracle
  - He evaluates the success of the test cases on the basis of the requirements and of his personal judgement
    - *–For example, in acceptance testing*
- Software oracle
  - An Oracle is another software having the same behavior of the software to be tested
    - *–For example, a known version of bubblesort can be used to test a faster quicksort*
    - *–A previous version of a software can be the oracle to evaluate the behavior of a newer software offering the same functionality (regression testing)*
- Implicit oracle
  - Testing against crashes, the oracle found a failure in each case a crash occurs
- Formal specification oracle
  - If the requirements are expressed in a formal way, then the oracle can be automatically derived from them

What is the role of the comparator?

- If the oracle provides exact expected values for the output, the comparator has to evaluate a simple bit comparison
- If the oracle is expressed in terms of a set of valid values, the comparator has to evaluate if the obtained result belongs to this set
  - For example, the expected behavior can be expressed as «a positive number», then the comparator has to evaluate a «greater than» condition
- If the oracle is expressed in terms that are not exactly represented by the computer, then the comparator has to perform an approximate evaluation
  - If the expected behavior is the number PI, then the comparator has to compare the difference between the obtained result and an approximate representation of PI
  - If the expected behavior is an image of Naples, the comparator has to compare the output image with a base of Naples images in order to argue if the image is sufficiently similar to a Naples image